MPSI - ITC - TD 16 : Complément à deux

Représentation des écritures binaire dans ce TP

Contrairement au TP précédent, dans ce TP les écritures binaires seront des listes de 0 et de 1 avec **les chiffres qui correspondent à des puissances de** 2 **élevées à gauche de la liste** (dit autrement, les indices élevés dans l'écriture binaire telle que formalisée en cours correspondent aux indices faibles pour la liste python qui la représente) **et les chiffres qui correspondent à des puissances de** 2 **faibles à droite de la liste** (dit autrement, les indices faibles dans l'écriture binaire telle que formalisée en cours correspondent aux indices élevés de la liste python qui la représente). Cela se rapproche de la façon qu'on utilise pour représenter les écritures binaires au tableau ou sur une feuille.
Par exemple, si on encode le nombre 3 en complément à deux sur 5 chiffres, cela donne `00011` qui sera représenté par la liste python `[0, 0, 0, 1, 1]`.

Déclencher une erreur quand une hypothèse n'est pas respectée

Quand on écrit une fonction, on fait souvent des hypothèses sur les données qu'on va recevoir en entrée. Ces hypothèses sont appelées des *pré-conditions*.
Quand une pré-condition d'une fonction n'est pas respectée, on peut vouloir que la fonction déclenche une erreur. En python, on utilise le mot-clef `assert`.

Exemple. Si on veut coder une fonction `factorielle` qui déclenche une erreur si son argument n'est pas positif ou nul, on peut procéder comme suit :

```
def factorielle(n):
    assert(n >= 0)
    if n < 2:
        return 1
    return n * factorielle(n-1)
```

Tester le déclenchement d'erreur avec doctest

On peut utiliser doctest pour vérifier que notre fonction déclenche bien une erreur quand elle est censée le faire. Pour cela, on précise juste la première ligne et la dernière ligne de l'erreur, avec trois petits points entre les deux.

Exemple

```
def factorielle(n):
    """
    Entrée
    ------
      Un entier positif ou nul.

    Sortie
    ------
      La factorielle de l'entier en entrée.

    Erreurs possibles
    -----------------
      Déclenche une erreur si l'entrée est négative.

    >>> factorielle(5)
    120

    >>> factorielle( -7 )
    Traceback (most recent call last):
    ...
    AssertionError
    """
    assert(n >= 0)
    if n < 2:
        return 1
    return n * factorielle(n-1)
```

Exercice 1 (Complémentaire)

Vous disposez d'un fichier prérempli pour vous faire gagner du temps (dans le navigateur, clic-droit sur le lien, enregistrer la cible du lien sous).

1. Écrire une fonction `complement_chiffre` qui prend en entrée un chiffre binaire (qui vaut soit 0 soit 1) et qui renvoie son complémentaire.
   Votre fonction déclenchera une erreur si l'entrée n'appartient pas à l'ensemble $\{0, 1\}$.
   Inspirez-vous de l'exemple fourni pour vos tests.
2. Écrire une fonction `complement_ecriture` qui prend en entrée une écriture binaire et renvoie son complémentaire.
   Votre fonction fera appel à la fonction `complement_chiffre`.

Exercice 2 (Encodage et décodage en complément à deux)

1. Écrire une fonction `encode_comp_2` qui prend en paramètre un entier relatif n et un nombre de chiffres k et encode n en complément à 2 sur k chiffres.

   Votre fonction déclenchera une erreur si k est strictement inférieur à 2, ou si n ne fait pas partie des nombres pouvant être encodés en complément à 2 sur k chiffres.

   Compléter les tests dans la docstring de votre fonction.
2. Écrire une fonction `decode_comp_2` qui prend en entrée une écriture binaire, et qui la décode, en l'interprétant comme du complément à 2.

   Votre fonction déclenchera une erreur si la taille de l'entrée est strictement inférieure à 2.

   Compléter les tests dans la docstring de votre fonction.

Exercice 3 (Bonus : Code de Gray)

On se place dans un contexte où le nombre de chiffres binaires à utiliser est fixé.
Pour certains algorithmes et pour certains appareils électroniques, on a besoin d'encoder les nombres positifs en binaire de telle sorte à ce que **quand on passe d'un nombre au suivant** (addition de 1) **il y ait au plus 1 chiffre binaire qui change à la fois**.
Un encodage binaire des nombres positifs qui respecte cette propriété est appelé un *code de Gray*.

Remarque : l'écriture binaire classique d'un nombre positif, vue en cours, n'est pas un code de Gray. Par exemple, sur 2 chiffres, l'écriture binaire de 1 est `01` et l'écriture binaire de 2 est `10`. On voit donc que lorsqu'on est passé de 1 à 2 (addition de 1), il y a deux chiffres qui ont changé au même temps.
On propose ci-dessous un algorithme d'encodage possible pour un code de Gray.

Encodage Soit n le nombre à encoder sur k bits. Soit $b_{k-1}...b_0$ son écriture binaire classique. On définit son code de Gray $g = g_{k-1}...g_0$ par

$$\begin{cases} g_{k-1} = b_{k-1} \\ \forall i \in [\![0, k-2]\!], g_i = b_i \text{ si } b_{i+1} = 0 \\ \forall i \in [\![0, k-2]\!], g_i = \overline{b_i} \text{ si } b_{i+1} = 1 \end{cases}$$

1. À la main, donner l'écriture binaire ainsi que le code de Gray des entiers entre 0 et 7.
2. Écrire une fonction `encode_gray(n, k)` qui renvoie le code de Gray de `n` sur k chiffres.
3. Vérifier que vous obtenez bien un code de Gray (par exemple sur 5 chiffres).
4. Écrire une fonction `decode_gray(g)` qui renvoie l'entier associé au code de Gray `g`.
5. Plus tôt dans l'année on a déjà manipulé un code de Gray, sans le dire. Sauriez-vous dire quand ? Était-ce le même code de Gray, ou un code de Gray différent ?